

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по учебной практике**  
**Тема: Проект**  
**Будильник погоды**

Студент гр. 8304	_____	Мухин А.М.
Студент гр. 8304	_____	Бочаров Ф.Д.
Руководитель	_____	Фирсов М.А.

Санкт-Петербург  
2020

## **ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ**

Студент Мухин А.М группы 8304

Студент Бочаров Ф.Д. группы 8304

Тема практики: Будильник погоды

Задание на практику:

Разработать программу позволяющую пользователю устанавливать будильник на необходимое время и получать актуальную погоду на это время. Об изменениях пользоваться должен получать уведомление.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 08.07.2020

Дата защиты отчета: 12.07.2020

Студент	_____	Бочаров Ф.Д.
Студент	_____	Мухин А.М.
Руководитель	_____	Фирсов М.А.

## **АННОТАЦИЯ**

Целью работы является получения навыков работы с такой парадигмой программирования, как объектно-ориентированное программирование. Для получения данных знаний выполняется один из вариантов мини-проекта. В процессе выполнения мини-проекта необходимо реализовать графический интерфейс к данной задаче, организовать ввод и вывод данных с его помощью, реализовать сам алгоритм, научиться работать в команде. В данной работе в качестве мини-проекта была решена проблема встречи с друзьями в плохую погоду. Пользователь может ввести город, и поставить будильник на определенный момент времени, как только в этот момент прогноз покажет дождь, будильник сработает и оповестит пользователя и плохой погоде и возможной отмене встречи с друзьями. Также при разработке выполняется написание тестирования, для проверки корректности алгоритма и работы интерфейса.

## **SUMMARY**

The aim of the work is to gain skills in working with such a programming paradigm as object-oriented programming. To obtain this knowledge, one of the mini-project options is performed. In the process of implementing a mini-project, it is necessary to implement a graphical interface to this task, organize the input and output of data with its help, implement the algorithm itself, learn how to work in a team. In this work, as a mini-project, the problem of meeting friends in bad weather was solved. The user can enter the city, and set the alarm for a certain point in time, as soon as at this moment the forecast shows rain, the alarm will work and will notify the user of bad weather and the possible cancellation of a meeting with friends. Also during development, testing is written to verify the correctness of the algorithm and the operation of the interface.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1. ТРЕБОВАНИЯ К ПРОГРАММЕ.....	6
1.1. Исходные требования к программе .....	6
1.1.1 Требования к вводу исходных данных .....	6
1.1.2 Требования к визуализации .....	6
1.1.3 Требования к выходным данным .....	7
1.2. Уточнение требований после первого этапа.....	8
1.2.1 Требования к вводу исходных данных .....	8
1.2.2 Требования к визуализации .....	8
1.2.3 Требования к выходным данным .....	9
1.3. Уточнение требований второго этапа.....	11
1.3.1 Требования к вводу исходных данных .....	11
1.3.2 Требования к визуализации .....	11
1.3.3 Требования к выходным данным .....	11
2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЯ РОЛЕЙ В БРИГАДЕ .....	12
2.1. План разработки.....	12
2.2. Распределение ролей в бригаде.....	13
3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ .....	14
3.1. Структуры данных и основные методы .....	14
4. ТЕСТИРОВАНИЕ .....	16
4.1 Мануальное тестирование программы .....	16
4.2. Unit тестирование программы.....	17
ЗАКЛЮЧЕНИЕ .....	19
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	20
ПРИЛОЖЕНИЕ А. UML-ДИОГРАММА.....	21
ПРИЛОЖЕНИЕ Б. КОД ПРОГАММЫ .....	22
ПРИЛОЖЕНИЕ В. UNIT-ТЕСТИРОВАНИЕ .....	44

## **ВВЕДЕНИЕ**

Программа генерирует прогноз погоды, изображение которого визуализировано. Пользователь выбирает город и получает прогноз погоды на несколько дней вперед с 3-х часовым интервалом. Далее пользователь может выбрать желаемое время. После нажатия на кнопку “Поставить будильник” программа запоминает даты и в течении промежутка времени проверяет прогноз с некоторой периодичностью. Также пользователь может получить текущий прогноз погоды. Если погода испортилась, оповещает пользователя. Программа должна работать постоянно.

# 1. ТРЕБОВАНИЯ К ПРОГРАММЕ

## 1.1. Исходные требования к программе

### 1.1.1 Требования к вводу исходных данных

Для корректной работы алгоритма необходимо:

- Ввести существующий город.
- Нажать “Текущая погода”.

### 1.1.2 Требования к визуализации

Прототип программы представлен на данных рисунках.

00:00					
03:00					
06:00					
09:00					
12:00					
15:00					
18:00					
21:00					

Рисунок 1

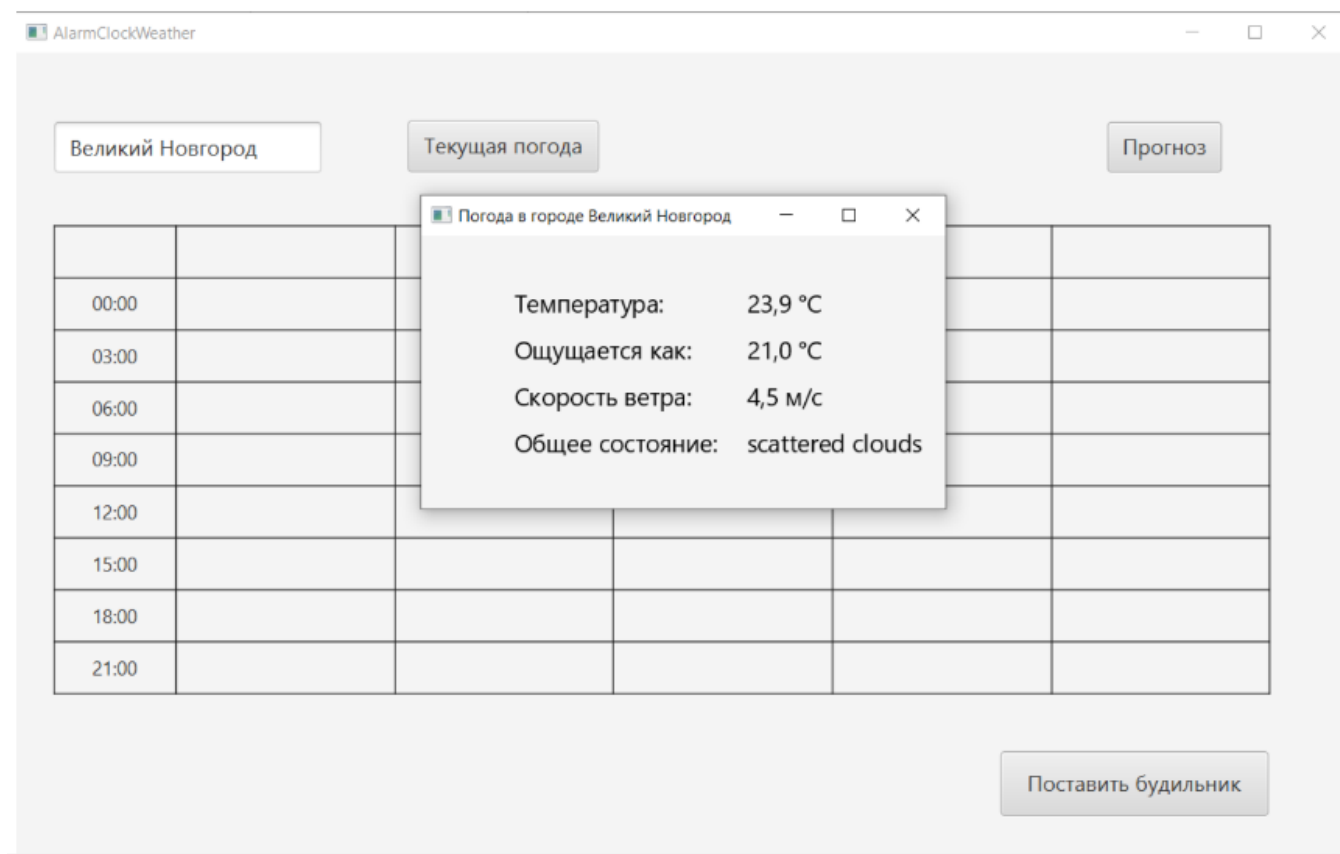


Рисунок 2

На рисунке 1, пользователь вводит название города и нажимает кнопку “Прогноз”.

На рисунке 2, всплывающее окно после нажатия “Текущий прогноз”.

### 1.1.3 Требования к выходным данным

Выходными данными является оповещение пользователя в случае изменения погодных условий на неблагоприятные.

## 1.2. Уточнение требований после первого этапа

### 1.2.1 Требования к вводу исходных данных

Для корректной работы алгоритма необходимо:

- Ввести существующий город.
- Выбрать нужную дату и нажать “Поставить будильник”.

### 1.2.2 Требования к визуализации

Прототип программы представлен на данных рисунках.



Рисунок 3





Рисунок 4

На рисунке 3, прогноз погоды по выбранному городу.

На рисунке 4, выбранные пользователями будильники.

### 1.2.3 Требования к выходным данным

Выходными данными является оповещение пользователя в случае изменения погодных условий на неблагоприятные.

## **2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЯ РОЛЕЙ В БРИГАДЕ**

### **2.1. План разработки**

1. К 29 июня 2020 года выбрать тему мини-проекта, распределить роли между участниками бригады, создать примерный план работы.
2. Ко 2 июля 2020 года, выполнить первую итерацию, то есть создать прототип программы, прототип визуального дизайна, а также сделать отчёт по первой итерации.
3. К 3 июля 2020 года исправить все недочеты, обнаруженные при защите на первой итерации.
4. К 4 июля 2020 года реализовать генерацию лабиринта.
5. К 4 июля 2020 года реализовать качественный дизайн интерфейса для программы.
6. К 4 июля 2020 года частично выполнить первые два раздела итогового отчёта по летней практике.
7. К 5 июля 2020 года провести тестирование генерации лабиринта.
8. К 6 июля 2020 года реализовать работу кнопок “Текущая погода”
9. К 6 июля провести тестирование алгоритма будильника.
10. 7 июля провести разбор недочётов после защиты 2-ой итерации
11. К 9 июля 2020 года завершить итоговый отчёт по летней практике.
12. К 10 июля провести улучшение дизайна итоговой программы.

## 2.2. Распределение ролей в бригаде

- Мухин А.М.
  - Алгоритм будильника
  - Все побочные вещи, связанные с кодом
- Бочаров Ф.Д.
  - Тестирование
  - Фронтенд

### 3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

#### 3.1. Структуры данных и основные методы

Класс **Controller.java** реализует главное окно.

Методы:

`void initialize()` – заполнение лэйблов погодой.

`private void fillTable(String information)`

`private int fillColumn(int columnIndex, int handlingElement, JSONArray information)` - очищения столбцов и строк

Класс **CurrentForecastController.java** реализует всплывающее окно Текущей погоды.

Методы:

`void getInformation() throws IOException, ParseException` – получение и вывод текущей погоды

Класс **Web.java** запрашивает погоду и загружает изображение условий.

Методы:

`public static void saveImage(String imageUrl, String destinationFile) throws IOException`

- сохранение изображения погодных условий.

`public static String requestWeather(TypeRequest type, String city) throws IOException` - запрос погоды.

## 4. ТЕСТИРОВАНИЕ

### 4.1 Мануальное тестирование программы

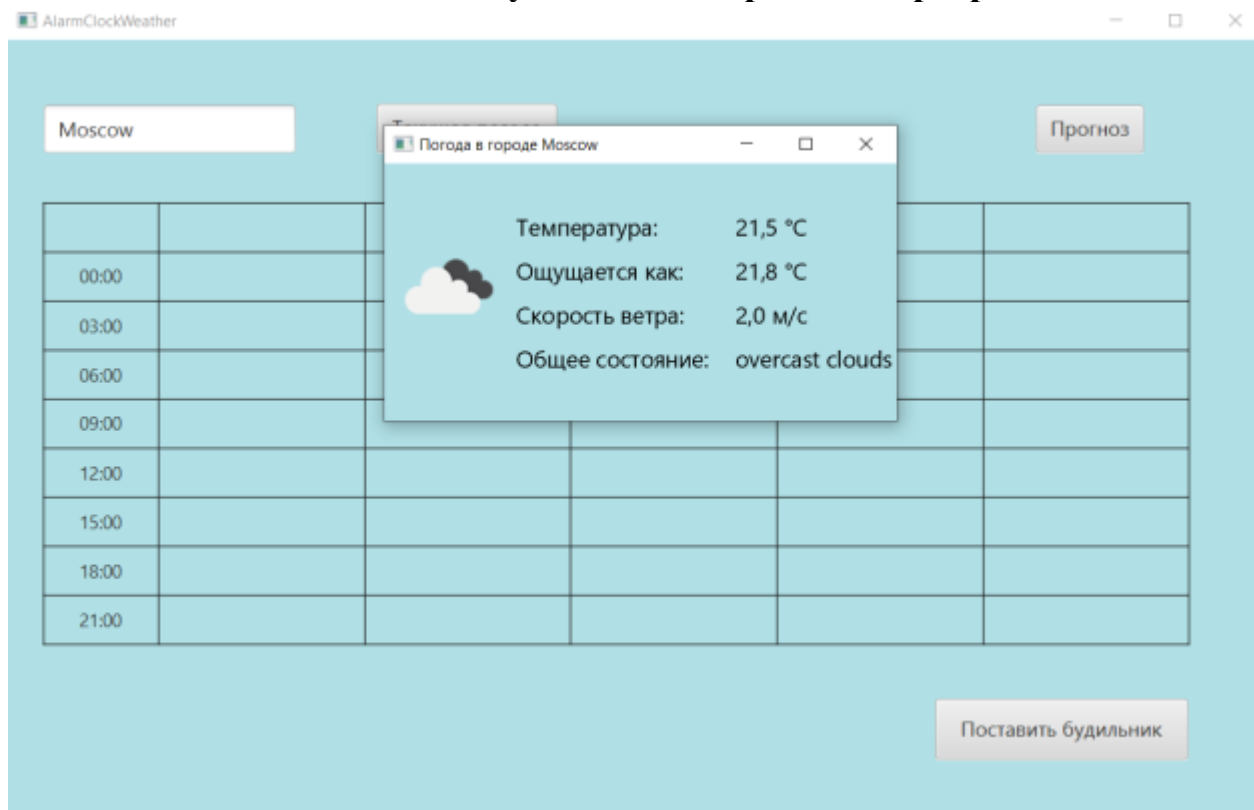


Рисунок 5 "Текущая погода"

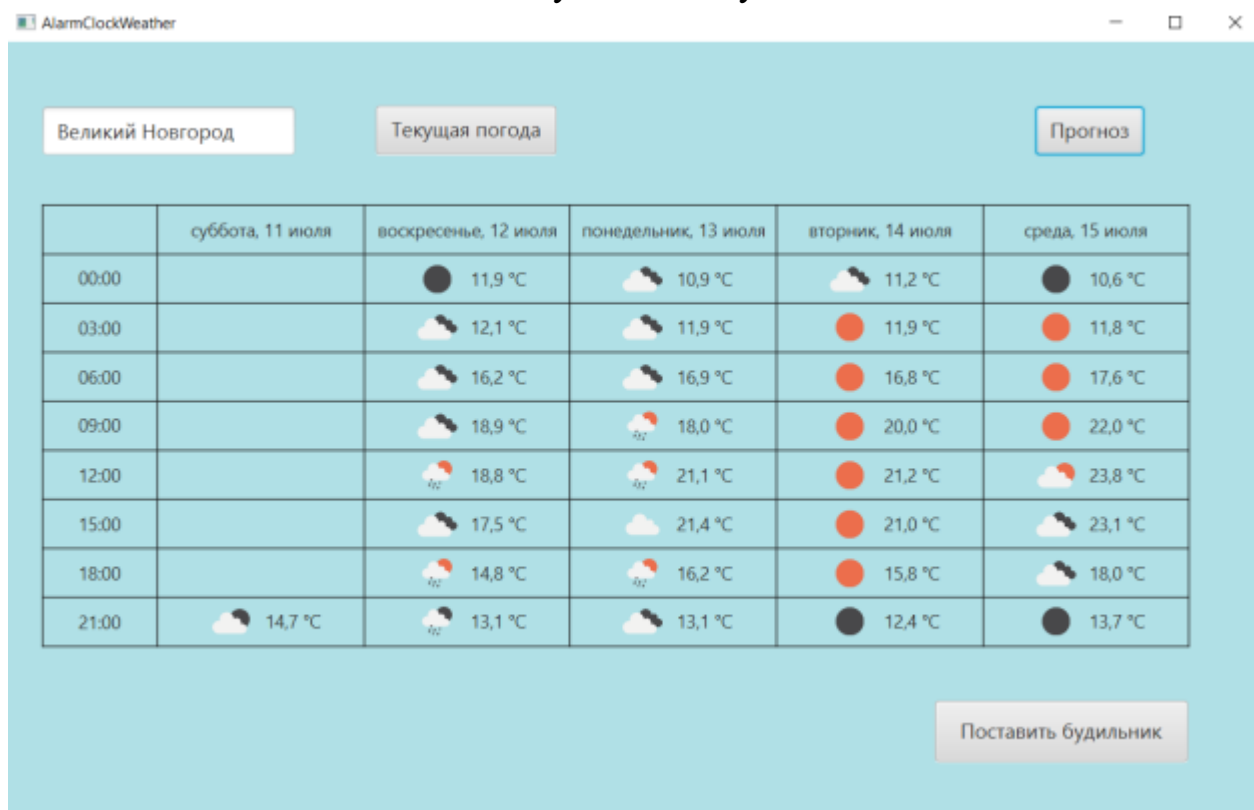


Рисунок 6 "Прогноз погоды на ближайшее время"

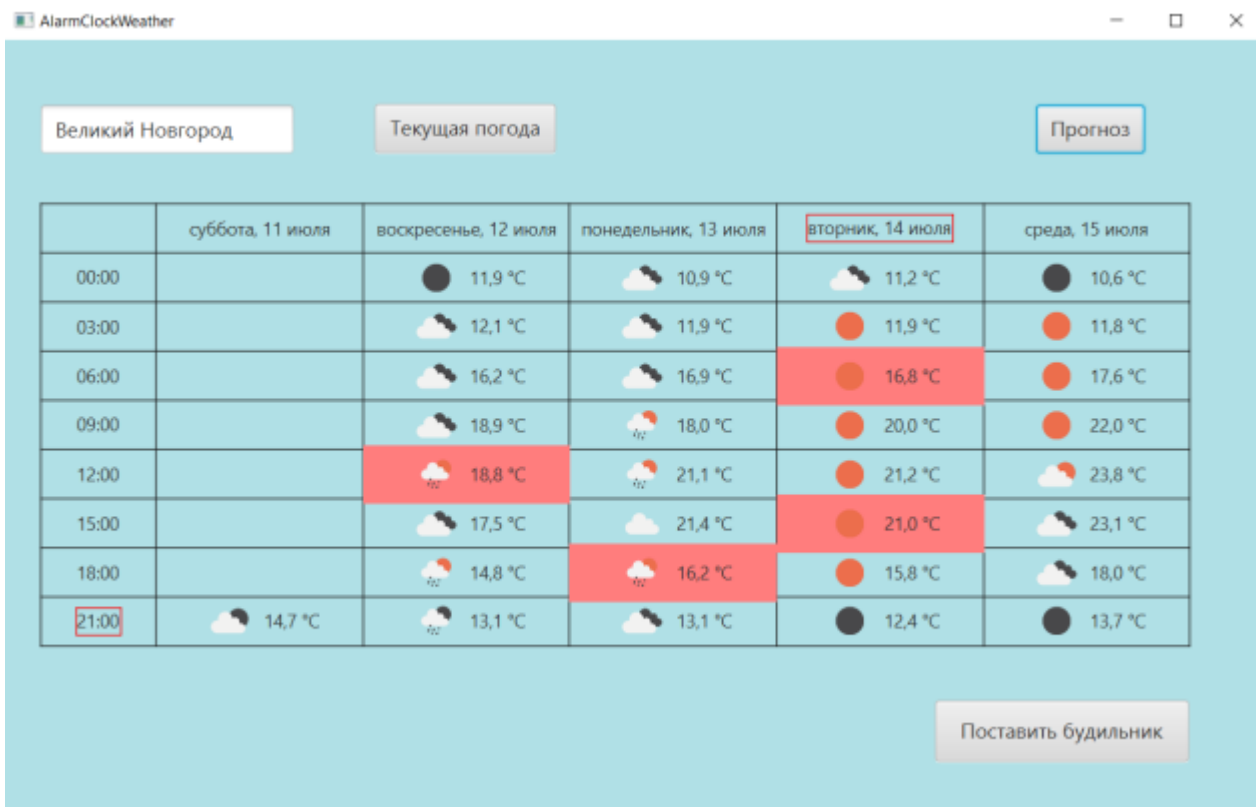


Рисунок 7 "Установка будильников"

## 4.2. Unit тестирование программы

Для проведение автоматического тестирования программы использовалась библиотека JUnit5. UNIT тесты были написаны с целью проверить web запросы к используемому API погоды.

`checkForWrongWebRequest()` – метод, который проверяет, что веб запрос вернул null.

`checkForCorrectWebRequest()` – метод, который проверяет, что веб запрос возвращает не null.

JVM. UNIT тесты представлены в Приложении В.

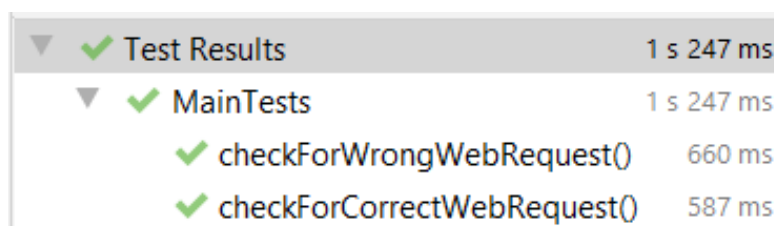


Рисунок 8 "Корректное выполнение unit тестов"

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы было спроектирована и разработана программа для оповещения пользователя об изменении погоды на выбранное время. Также была визуализирована работа алгоритма, и создан интерфейс позволяющий управлять работой программы. В ходе выполнения проекта были добавлены следующие возможности: возможность просмотра текущей работы в городе.

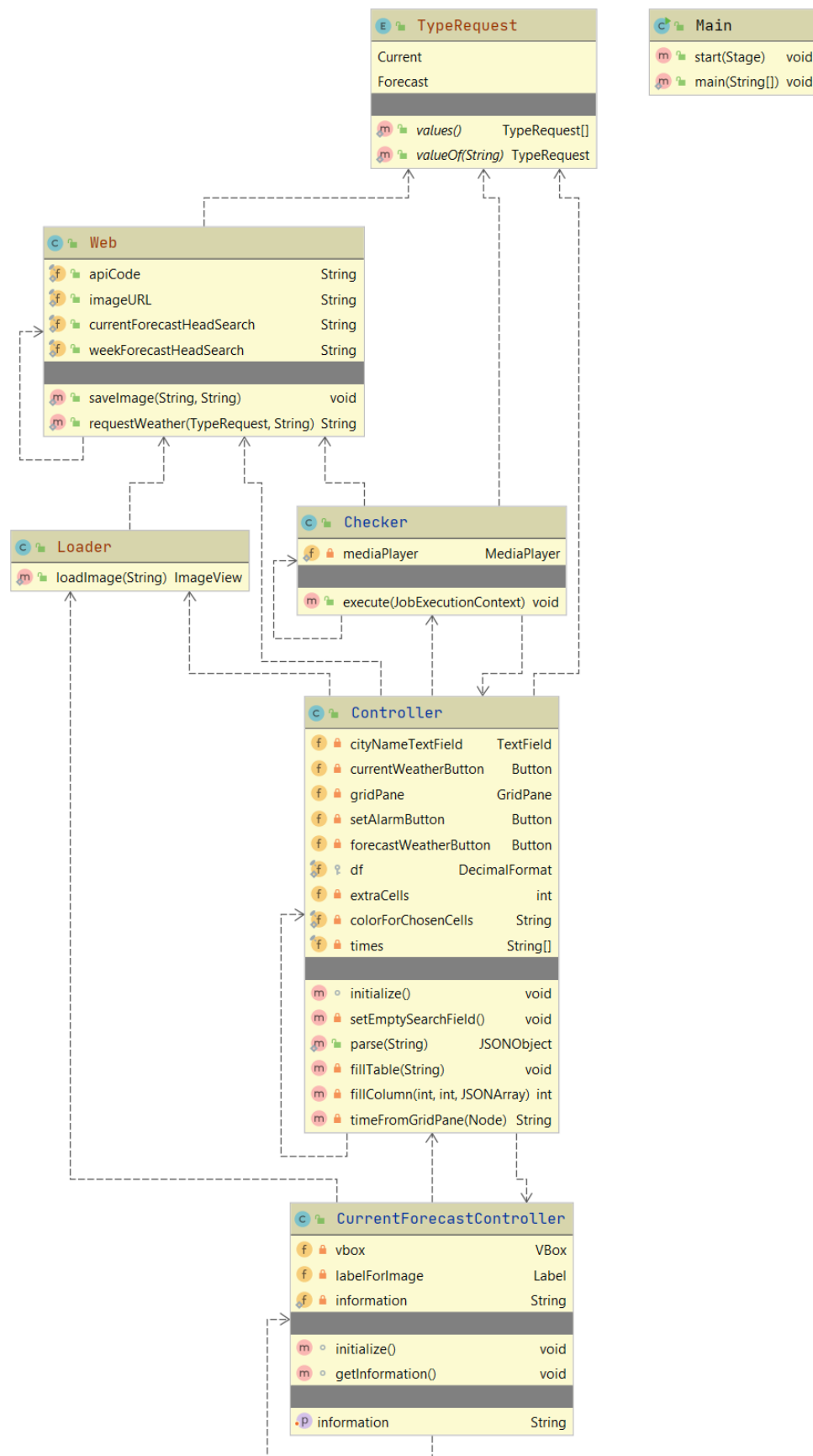
Были получены навыки программирования в JavaFX, а также парсинг с помощью библиотеки JSON.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Учебный курс по основам Java на Stepik: <https://stepik.org/course/187/>
2. Официальная документация к Java: <https://docs.oracle.com/en/java/javase/>
3. Java 8. Руководство для начинающих. Герберт Шилдт
4. <https://ru.wikipedia.org/wiki/>
5. <https://habr.com/ru/>
6. <https://ru.stackoverflow.com/>



## ПРИЛОЖЕНИЕ А. UML-ДИОГРАММА



## ПРИЛОЖЕНИЕ Б. КОД ПРОГРАММЫ

### Файл Controller.java

Package

controllers;

```
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.geometry.HPos;
import javafx.geometry.Pos;
import javafx.scene.Cursor;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import jobs.Checker;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import org.quartz.*;
import org.quartz.impl.StdSchedulerFactory;
import web.Loader;
import web.TypeRequest;
import web.Web;

import java.io.IOException;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Calendar;
import java.util.*;

import static org.quartz.TriggerBuilder.newTrigger;
```

```

public class Controller {
    @FXML
    private TextField cityNameTextField;

    @FXML
    private Button currentWeatherButton;

    @FXML
    private GridPane gridPane;

    @FXML
    private Button setAlarmButton;

    @FXML
    private Button forecastWeatherButton;

    protected static final DecimalFormat decimalFormat = new DecimalFormat("0.0");
    private int extraCells = 0;
    private static final String colorForChosenCells = "#ff7d7d";
    private final String[] times = {"00:00", "03:00", "06:00", "09:00", "12:00",
    "15:00", "18:00", "21:00"};

    @FXML
    void initialize() {
        /* заполняем столбцы со временем */
        for (int i = 1; i < 9; i++) {
            Label label = new Label(times[i-1]);
            label.setFont(Font.font(14));
            GridPane.setHalignment(label, HPos.CENTER);
            gridPane.add(label, 0, i);
        }

        /* если нажимаем на поле для ввода города, красим его рамку в стандартный цвет
        */
        cityNameTextField.setOnMouseClicked(mouseEvent -> cityNameTextField.setStyle("-
fx-border-color: default;"));

        /* при нажатии на кнопку "Текущая погода" */

```

```

currentWeatherButton.setOnAction(value -> {
    /* получаем информацию с weatherMapAPI */
    String information = null;
    try {
        information = Web.requestWeather(TypeRequest.Current,
cityNameTextField.getText());
    } catch (IOException e) {
        e.printStackTrace();
    }

    /* если запрос успешен, отображаем маленькое окно с текущей погодой */
    if (information != null) {
        FXMLLoader page = new
FXMLLoader(getClass().getResource("../fxml\\currentForecast.fxml"));
        Parent root;
        CurrentForecastController controller;
        try {
            root = page.load();
            controller = page.getController();
            controller.setInformation(information);
            Stage stage = new Stage();
            stage.setTitle("Погода в городе " + cityNameTextField.getText());
            stage.setScene(new Scene(root));
            stage.showAndWait();
        } catch (IOException | ParseException e) {
            e.printStackTrace();
        }
        /* если запрос вернул null, значит либо неправильно введено название
города, либо поле пустое, либо нет интернета */
    } else {
        setEmptySearchField();
    }
});

/* при нажатии на кнопку "Поставить будильник" */
setAlarmButton.setOnAction(actionEvent -> {
    List<String> times = new ArrayList<>();
    String tmpTime;
    Calendar scheduleTime = null;
    Date currentTime = null;
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    /* проходим по сетке и выбираем ячейки, которые выделены цветом */
    for (Node node : gridPane.getChildren()) {

```

```

        if (node.getStyle().contains("-fx-background-color: " +
colorForChosenCells)) {
            /* возвращаем выделенной ячейке её родной цвет */
            node.setStyle("-fx-background-color: default");
            tmpTime = timeFromGridPane(node);

            currentTime = new Date();
            scheduleTime = new GregorianCalendar();

            scheduleTime.roll(Calendar.DAY_OF_MONTH,
GridPane.getColumnIndex(node)-1);

            scheduleTime.set(Calendar.HOUR_OF_DAY,
Integer.parseInt(tmpTime.substring(0, 2)));
            scheduleTime.set(Calendar.MINUTE,
Integer.parseInt(tmpTime.substring(3, 5)));
            scheduleTime.set(Calendar.SECOND, 0);
            /* сравниваем выбранное время в ячейке и текущее, если текущее
время больше чем выбранное, то добавляем выбранное в массив*/
            if (scheduleTime.getTime().after(currentTime)) {
                times.add(dateFormat.format(scheduleTime.getTime()));
            }
        }
    }
    /* если массив времени не пустой */
    if (!times.isEmpty()) {
        /* создаём расписание, которое будет чекать дождь на выбранное время */
        SchedulerFactory schedulerFactory = new StdSchedulerFactory();
        Scheduler scheduler;
        long diffInMillies;
        long diff;
        SimpleTrigger trigger;
        try {
            /* указываем расписанию откуда брать код для выполнения */
            JobBuilder jobBuilder = JobBuilder.newJob(Checker.class);
            for (String time : times) {
                scheduler = schedulerFactory.getScheduler();
                scheduler.start();
                String scheduleName = time +
cityNameTextField.getText().replace(' ', '+');
                JobDetail job = jobBuilder.withIdentity(scheduleName, "all
jobs").build();

                job.getJobDataMap().put("scheduler", scheduler);
                /* вычисляем, сколько часов проходит между текущим моментом и

```

```

выбранной датой */
        diffInMillies = Math.abs(currentTime.getTime() -
scheduleTime.getTime().getTime());
        diff = diffInMillies / 3600000;

        trigger = newTrigger()
            .withIdentity(scheduleName)
            .startNow()
            /* говорим расписанию, сколько надо будет выполнить раз
код и с каким диапазоном */

.withSchedule(SimpleScheduleBuilder.repeatHourlyForTotalCount((int) diff/3, 3))
            .build();
            /* запускаем выполнение */
            scheduler.scheduleJob(job, trigger);
        }
    } catch (SchedulerException e) {
        e.printStackTrace();
    }
}
});

/* при нажатии на кнопку "Прогноз" */
forecastWeatherButton.setOnAction(value -> {
    String information = null;
    try {
        information =
Web.requestWeather(TypeRequest.Current, cityNameTextField.getText());
    } catch (IOException e) {
        e.printStackTrace();
    }

    if (information != null) {
        try {
            String response = Web.requestWeather(TypeRequest.Forecast,
cityNameTextField.getText());
            fillTable(response);
        } catch (IOException | ParseException | InterruptedException e) {
            e.printStackTrace();
        }
    } else {
        setEmptySearchField();
    }
});

```

```

    }

    /* если в поле для ввода города нет текста, то красим его рамку в красный цвет и
    пишем информативное сообщение */
    private void setEmptySearchField() {
        cityNameTextField.setStyle("-fx-text-box-border: red; -fx-focus-color: red;");
        cityNameTextField.setText("");
        cityNameTextField.setPromptText("Введите город!");
    }

    public static JSONObject parse(String information) throws ParseException {
        JSONParser parser = new JSONParser();
        return (JSONObject) parser.parse(information);
    }

    private void fillTable(String information) throws ParseException, IOException,
    InterruptedException {
        /* отчищаем таблицу от старых значений */
        gridPane.getChildren().removeIf(node -> GridPane.getColumnIndex(node) != null
        && GridPane.getColumnIndex(node) != 0);
        JSONObject json = parse(information);
        JSONArray list = (JSONArray) json.get("list");
        /* счётчик по элементам JSONArray list */
        int nextHandlingElement = 0;

        /* заполняем столбы сетки, каждый раз будем возвращать индекс элемента в
        JSONArray list, на котором мы остановились */
        for (int i = 1; i < 6; ++i) {
            nextHandlingElement = fillColumn(i, nextHandlingElement, list);
        }
    }

    private int fillColumn(int columnIndex, int handlingElement, JSONArray information)
    throws IOException, InterruptedException {
        int startRow;
        double temperature;
        DateTimeFormatter formatterRead = DateTimeFormatter.ofPattern("yyyy-MM-dd
        HH:mm:ss");
        DateTimeFormatter formatterWrite = DateTimeFormatter.ofPattern("EEEE, dd
        MMMM");
        JSONObject period = (JSONObject) information.get(handlingElement++);

```

```

        /* если эта первая колонка, то возможна ситуация, что начнём заполнять её не с
        начала */
        if (columnIndex == 1) {
            String dtText = (String) period.get("dt_txt");
            LocalDateTime dateTime = LocalDateTime.parse(dtText, formatterRead);
            /* считаем индекс, с которого надо надо заполнять сетку */
            startRow = dateTime.getHour() / 3;
            extraCells = startRow;
        } else {
            startRow = 0;
        }

        /* ставим дату в самую верхнюю ячейку столбца */
        String dtText = (String) period.get("dt_txt");
        LocalDateTime dateTime = LocalDateTime.parse(dtText, formatterRead);
        Label label = new Label(dateTime.format(formatterWrite));
        label.setFont(Font.font(14));
        GridPane.setHalignment(label, HPos.CENTER);
        gridPane.add(label, columnIndex, 0);

        /* проходимся по всему столбику и заполняем его значениями температуры и
        картиночками */
        for (int i = startRow + 1; i < 9; ++i) {
            try {
                temperature = (long) ((JSONObject) period.get("main")).get("temp");
            } catch (ClassCastException e) {
                temperature = (double) ((JSONObject) period.get("main")).get("temp");
            }
            Label text = new Label(decimalFormat.format(temperature - 273) + " °C");
            text.setFont(Font.font(14));
            text.setMaxWidth(Double.MAX_VALUE);
            text.setMaxHeight(Double.MAX_VALUE);
            text.setAlignment(Pos.CENTER);
            JSONArray weather = (JSONArray) period.get("weather");
            JSONObject elemInWeather = (JSONObject) weather.get(0);
            String imageName = elemInWeather.get("icon") + "@2x.png";

            ImageView imageView = Loader.loadImage(imageName);
            imageView.setFitHeight(45);
            imageView.setFitWidth(45);
            text.setGraphic(imageView);
            /* устанавливаем изменения на вхождение и выход курсора из ячейки, клик по
            ячейке и чтобы появлялась ручка при наведении на ячейку */

```



```

        text.setOnMouseEntered(mouseEvent -> {
            int row = GridPane.getRowIndex(text);
            int column = GridPane.getColumnIndex(text);
            gridPane.getChildren().get(row).setStyle("-fx-border-color: red");
            if (column == 1) {
                gridPane.getChildren().get(column*9).setStyle("-fx-border-color:
red");
            } else {
                gridPane.getChildren().get(column*9 - extraCells).setStyle("-fx-
border-color: red");
            }
        });
        text.setOnMouseExited(mouseEvent -> {
            int row = GridPane.getRowIndex(text);
            int column = GridPane.getColumnIndex(text);
            gridPane.getChildren().get(row).setStyle("-fx-border-color: default");
            if (column == 1) {
                gridPane.getChildren().get(column*9).setStyle("-fx-border-color:
default");
            } else {
                gridPane.getChildren().get(column*9 - extraCells).setStyle("-fx-
border-color: default");
            }
        });
        text.setCursor(Cursor.HAND);
        text.setOnMouseClicked(mouseEvent -> {
            if (text.getStyle().contains(colorForChosenCells)) {
                text.setStyle("-fx-background-color: default");
            } else {
                text.setStyle("-fx-background-color: " + colorForChosenCells);
            }
        });
    });

    period = (JSONObject) information.get(handlingElement++);
    gridPane.add(text, columnIndex, i);
}

return handlingElement - 1;
}

/* по ячейки в сетке, получаем время, к которому она относится */
private String timeFromGridPane(Node node) {
    return ((Label)
gridPane.getChildren().get(GridPane.getRowIndex(node))).getText();
}

```

```
}  
}
```

## Файл CurrentForecastController.java

```
package
```

```
controllers;
```

```
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import javafx.scene.image.ImageView;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Text;  
import org.json.simple.JSONArray;  
import org.json.simple.JSONObject;  
import org.json.simple.parser.ParseException;  
import web.Loader;
```

```
import java.io.IOException;
```

```
public class CurrentForecastController {
```

```
    @FXML
```

```
    private VBox vbox;
```

```
    @FXML
```

```
    private Label labelForImage;
```

```
    private static String information;
```

```
    @FXML
```

```
    void initialize() {}
```

```
    void getInformation() throws IOException, ParseException {
```

```
        JSONObject json = Controller.parse(information);
```

```
        JSONObject main = (JSONObject) json.get("main");
```

```
        JSONArray weather = (JSONArray) json.get("weather");
```

```
        JSONObject wind = (JSONObject) json.get("wind");
```

```
        double temperature = (double) main.get("temp") - 273;
```

```
        double feelsLike = (double) main.get("feels_like") - 273;
```

```
        JSONObject commonWeather = (JSONObject) weather.get(0);
```

```

String summary = (String) commonWeather.get("description");
double speed;

try {
    speed = (long) wind.get("speed");
} catch (ClassCastException e) {
    speed = (double) wind.get("speed");
}

/* загружаем картинку и устанавливаем в 4 поля соответствующие значения */
String imageName = commonWeather.get("icon") + "@2x.png";
ImageView image = Loader.loadImage(imageName);
labelForImage.setGraphic(image);

((Text)
(vbox.getChildren().get(0))).setText(Controller.decimalFormat.format(temperature) + "
°C");
((Text)
(vbox.getChildren().get(1))).setText(Controller.decimalFormat.format(feelsLike) + "
°C");
((Text)
(vbox.getChildren().get(2))).setText(Controller.decimalFormat.format(speed) + " м/с");
((Text) (vbox.getChildren().get(3))).setText(summary);
}

public void setInformation(String information) throws IOException, ParseException {
    CurrentForecastController.information = information;
    getInformation();
}
}

```

## Файл Web.java

```

package
web;

```

```

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;

```

```

public class Web {
    public final static String apiCode = "67ff01ef1bd158284e098eba0512fb5d";
    final public static String imageURL = "http://openweathermap.org/img/wn/";
    public final static String currentForecastHeadSearch =
"http://api.openweathermap.org/data/2.5/weather?q=";
    public final static String weekForecastHeadSearch =
"http://api.openweathermap.org/data/2.5/forecast?q=";

    public static void saveImage(String imageUrl, String destinationFile) throws IOException
    {
        URL url = new URL(imageUrl);
        InputStream input = url.openStream();
        OutputStream output = new FileOutputStream(destinationFile);

        byte[] b = new byte[2048];
        int length;

        while ((length = input.read(b)) != -1) {
            output.write(b, 0, length);
        }

        input.close();
        output.close();
    }

    public static String requestWeather(TypeRequest type, String city) throws IOException {
        String mainStringRequest = null;
        if (type == TypeRequest.Current) {
            mainStringRequest = currentForecastHeadSearch;
        } else if (type == TypeRequest.Forecast) {
            mainStringRequest = weekForecastHeadSearch;
        }

        URL request = new URL(mainStringRequest + city.replace(' ', '+') + "&appid=" +
apiCode);
        HttpURLConnection connection = (HttpURLConnection) request.openConnection();
        connection.setRequestMethod("GET");
    }
}

```

```
        if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {  
            BufferedReader input = new BufferedReader(new  
InputStreamReader(connection.getInputStream()));  
            String response = input.readLine();  
            input.close();  
            return response;  
        } else {  
            return null;  
        }  
    }  
}
```